Standard Code Library

FLself

SCUT

August 24, 2022

Contents

一切的开始 **2**
一些宏定义 2

数据结构 **3**
ST 表 3
Fenwick Tree(树状数组) 4
线段树 4

数学 **5**
数论 5
数论整数 5
拉格朗日差值法 6
欧几里得 6
中国剩余定理 6
各种筛 7
多项式 10
组合数学 14
斯特林数 14
线性代数 14
线性基 14
博弈论 15

图论 **15**
LCA 15
连通性 16
tarjan 16
树 17
虚树 17
. . . . 18

计算几何 **18**
二维几何：点与向量 18

字符串 **19**
KMP 19
后缀自动机 19

杂项 **20**
STL 20

一切的开始

一些宏定义

- 需要 C++11

```cpp
// #define DEBUG
// #define InTerminal
// #define std_cpp17
#include<bits/stdc++.h>
#define int long long
#define PII std::pair<int, int>
#define VI std::vector<int>
#define VPII std::vector<std::pair<int, int> >
#define VVI std::vector<std::vector<int> >
#define ALL(a) (a).begin(), (a).end()
#define SIZ(a) ((int)(a).size())
#define FOR(i, l, r) for (int i = (l); i <= (r); ++i)
#define REP(i, r, l) for (int i = (r); i >= (l); --i)
#define lowbit(x) ((x) & (-(x)))
#define lbpos(x) (__builtin_ctz(x))
#define hbpos(x) (31 - __builtin_clz(x))

template<typename S, typename T> std::istream &operator>>(std::istream &is, std::pair<S, T> &pp) { is >> pp.first >> pp.second; return is; }
template<typename S, typename T> std::ostream &operator<<(std::ostream &os, std::pair<S, T> pp) { os << "(" << pp.first << ", " << pp.second << ")"; return os; }
template<typename S, std::size_t _siz> std::istream &operator>>(std::istream &is, std::array<S, _siz> &arr) { for (auto &x: arr) is >> x; return is; }
template<typename S, std::size_t _siz> std::ostream &operator<<(std::ostream &os, std::array<S, _siz> arr) { os << "("; for (auto x: arr) os << x << ", "; os << ")"; return os; }
template<typename T> std::istream &operator>>(std::istream &is, std::vector<T> &vec) { for (auto &x: vec) is >> x; return is;}
template<typename T> std::ostream &operator<<(std::ostream &os, const std::vector<T> &vec) { os << '{'; for (auto &x: vec) os << x << ", "; return os << "}";}
#ifdef std_cpp17
template<class Tuple, std::size_t... Is> void print_tuple_impl(std::ostream &os, const Tuple &t, std::index_sequence<Is...>) { ((os << (Is == 0? "" : ", ") << std::get<Is>(t)), ...); }
template<class... Args> std::ostream &operator<<(std::ostream &os, const std::tuple<Args...> &t) { os << "("; print_tuple_impl(os, t, std::index_sequence_for<Args...>{}); return os << ")"; }
#endif
#ifdef DEBUG
#ifdef InTerminal
#define dbg(x...) do { std::cerr << "\033[32;1m" << #x << " -> "; err(x); } while (0)
void err() { std::cerr << "\033[39;0m" << std::endl; }
#else
#define dbg(x...) do { std::cerr  << #x << " -> "; err(x); } while (0)
void err() { std::cerr << std::endl; }
#endif
template<typename T, typename... A>
void err(T a, A... x) { std::cerr << a << ' '; err(x...); }
#else
#define dbg(...)
#endif

using namespace std;
const int maxn = 2e5 + 3;
const int INF = 0x3f3f3f3f3f3f3f3f;
const int mod = 998244353;
mt19937 RD(time(0));



void solv() {


    return ;
}

signed main() {
    // freopen("./data.in", "r", stdin);
    std::ios::sync_with_stdio(false), std::cin.tie(0), std::cout.tie(0);
    int beg__TT = clock();

    signed _ttt;
    cin >> _ttt;

    while(_ttt--)
        solv();

    #ifdef DEBUG
    std::cerr << "use : " << (clock() - beg__TT) << "ms\n";
    #endif
    return 0;
}
```

数据结构

ST 表

- 一维

```cpp
class Sparcetable {
    vector<vector<int> > st;
    int siz;
    bool MX_flg = 0;
    inline int renew(int x, int y) {
        if (MX_flg) return  max(x, y);
        return min(x, y);
    }
public:
    // 注意 bhpos(0) 返回-1
    bool (*comp)(int, int);
    Sparcetable():siz(maxn) {st.resize(hbpos(maxn - 1) + 1, std::vector<int> (maxn));}
    Sparcetable(const std::vector<int>& a, bool _MX_flg = 1): siz(a.size()), MX_flg(_MX_flg) {
        int n = a.size();
        st.resize(hbpos(n) + 1, vector<int> (n + 1));
        for (int i = 1; i <= n; ++i) st[0][i] = a[i];
        for (int i = 1; i <= hbpos(siz); ++i) {
            for (int j = 1; j + (1 << i) <= siz + 1; ++j) {
                st[i][j] = renew(st[i - 1][j], st[i - 1][j + (1 << (i - 1))]);
            }
        }
    }
    int query(int l, int r) {
        int len = hbpos(r - l + 1);
        return renew(st[len][l], st[len][r - (1 << len) + 1]);
    }
};
```

- 二维

```cpp
int f[10][10][maxn][maxn];
#define _highbit(x) (31 - __builtin_clz(x))
inline int calc(int x, int y, int xx, int yy, int p, int q) {
    return max(
        max(f[p][q][x][y], f[p][q][xx - (1 << p) + 1][yy - (1 << q) + 1]),
        max(f[p][q][xx - (1 << p) + 1][y], f[p][q][x][yy - (1 << q) + 1])
    );
}
void init() {
    for (int x = 0; x <= _highbit(n); ++x)
    for (int y = 0; y <= _highbit(m); ++y)
        for (int i = 0; i <= n - (1 << x); ++i)
        for (int j = 0; j <= m - (1 << y); ++j) {
            if (!x && !y) { f[x][y][i][j] = a[i][j]; continue; }
            f[x][y][i][j] = calc(
                i, j,
                i + (1 << x) - 1, j + (1 << y) - 1,
                max(x - 1, 0), max(y - 1, 0)
            );
        }
}
inline int get_max(int x, int y, int xx, int yy) {
    return calc(x, y, xx, yy, _highbit(xx - x + 1), _highbit(yy - y + 1));
}
```

Fenwick Tree(树状数组)

- 一维

```
template<typename T>
class FenwickT {
    int n;
    vector<T> tr;
public:
    FenwickT(int siz): tr(siz), n(siz) {}
    FenwickT(int siz, T ini): tr(siz, ini), n(siz) {}
    void add(int p, T x) {
        for (int i = p; i < n; i += i & (-i)) tr[i] += x;
    }
    T query(int p) {
        T ret = T(0);
        for (int i = p; i > 0; i -= i & (-i)) ret += tr[i];
        return ret;
    }
    T range_sum(int l, int r) {
        return (query(r) - query(l - 1));
    }
};
```

线段树

```
class SegT{
    struct Pt{
        int val, ls, rs, sl;
        Pt():val(0), ls(0), rs(0), sl(0) {}
        Pt(int v, int l, int r, int s): val(v), ls(l), rs(r), sl(s) {}
        ~Pt() {}
    };
    int renew(int x, int y) {
        return x + y;
    }
    int renewsl(int x, int y) {
        return x + y;
    }
    int renewqj(int x, int y, int l, int r) {
        return x + y * (r - l + 1);
    }
public:
    std::vector<Pt> tr;
    int cnt = 0; // 结点数目
    SegT() {cnt = 1; tr.emplace_back();} // 建一棵空树

    // 修改
    void add(int l, int r, int p, int ql, int qr, int x) {
        if (ql <= l && r <= qr) {
            tr[p].val = renewqj(tr[p].val, x, l, r);
            tr[p].sl = renewsl(tr[p].sl, x);
            return;
        }
        int mid = l + (r - l) / 2;
        if (tr[p].sl) {
            if (!tr[p].ls) tr[p].ls = cnt++;
            if (!tr[p].rs) tr[p].rs = cnt++;
            if (cnt >= tr.size()) tr.resize(cnt + 1);
            int ls = tr[p].ls, rs = tr[p].rs;
           tr[ls].val = renewqj(tr[ls].val, tr[p].sl, l, mid), tr[rs].val = renewqj(tr[rs].val, tr[p].sl, mid + 1, r);
            tr[ls].sl = renewsl(tr[ls].sl, tr[p].sl), tr[rs].sl = renewsl(tr[rs].sl, tr[p].sl);
            tr[p].sl = 0;
        }
```

```cpp
        if (ql <= mid) {
            if (!tr[p].ls) tr[p].ls = cnt++;
            if (cnt >= tr.size()) tr.resize(cnt + 1);
            add(l, mid, tr[p].ls, ql, qr, x);
        }
        if (qr > mid) {
            if (!tr[p].rs) tr[p].rs = cnt++;
            if (cnt >= tr.size()) tr.resize(cnt + 1);
            add(mid + 1, r, tr[p].rs, ql, qr, x);
        }
        tr[p].val = renew(tr[p].ls? tr[tr[p].ls].val: 0, tr[p].rs? tr[tr[p].rs].val: 0);
    };

    // 查询
    int query(int l, int r, int p, int ql, int qr) {
        if (ql <= l && r <= qr) return tr[p].val;
        int mid = l + (r - l) / 2;
        if (tr[p].sl) {
            if (!tr[p].ls) tr[p].ls = cnt++;
            if (!tr[p].rs) tr[p].rs = cnt++;
            if (cnt >= tr.size()) tr.resize(cnt + 1);
            int ls = tr[p].ls, rs = tr[p].rs;
          tr[ls].val = renewqj(tr[ls].val, tr[p].sl, l, mid), tr[rs].val = renewqj(tr[rs].val, tr[p].sl, mid + 1, r);
            tr[ls].sl = renewsl(tr[ls].sl, tr[p].sl), tr[rs].sl = renewsl(tr[rs].sl, tr[p].sl);
            tr[p].sl = 0;
        }
        int ret = 0;
        if (ql <= mid) {
            if (!tr[p].ls) tr[p].ls = cnt++;
            if (cnt >= tr.size()) tr.resize(cnt + 1);
            ret = renew(ret, query(l, mid, tr[p].ls, ql, qr));
        }
        if (qr > mid) {
            if (!tr[p].rs) tr[p].rs = cnt++;
            if (cnt >= tr.size()) tr.resize(cnt + 1);
            ret = renew(ret, query(mid + 1, r, tr[p].rs, ql, qr));
        }
        tr[p].val = renew(tr[p].ls? tr[tr[p].ls].val: 0, tr[p].rs? tr[tr[p].rs].val: 0);
        return ret;
    };
};
```

数学

数论

数论整数

```cpp
constexpr int P = 998244353;
// assume -P <= x < 2P
int norm(int x) {
    // x %= P;
    if (x < 0) { x += P; }
    if (x >= P) { x -= P; }
    return x;
}
template<typename E>
E power(E n, int k) {
    E ret = E(1);
    while (k) {
        if (k & 1) ret *= n;
        n *= n;
        k >>= 1;
    } return ret;
}
struct Z {
    int x;
    Z(int x = 0) : x(norm(x)) {}
    int val() const { return x; }
    Z operator-() const { return Z(norm(P - x)); }
    Z inv() const { assert(x != 0); return power(*this, P - 2); }
    Z &operator*=(const Z &rhs) { x = (long long)(x) * rhs.x % P; return *this; }
    Z &operator+=(const Z &rhs) { x = norm(x + rhs.x); return *this; }
    Z &operator-=(const Z &rhs) { x = norm(x - rhs.x); return *this; }
    Z &operator/=(const Z &rhs) { return *this *= rhs.inv(); }
    friend Z operator*(const Z &lhs, const Z &rhs) { Z res = lhs; res *= rhs; return res; }
    friend Z operator+(const Z &lhs, const Z &rhs) { Z res = lhs; res += rhs; return res; }
    friend Z operator-(const Z &lhs, const Z &rhs) { Z res = lhs; res -= rhs; return res; }
    friend Z operator/(const Z &lhs, const Z &rhs) { Z res = lhs; res /= rhs; return res; }
    friend std::istream &operator>>(std::istream &is, Z &a) { long long v; is >> v; a = Z(v); return is; }
    friend std::ostream &operator<<(std::ostream &os, const Z &a) { return os << a.val(); }
};
```

拉格朗日差值法

欧几里得

- 扩展欧几里得

```cpp
int exgcd(int a, int b, int& x, int& y) {
    if (a == 0) {
        x = 0, y = 1;
        return b;
    }
    int ret = exgcd(b % a, a, x, y), xx = x;
    x = y - b / a * x;
    y = xx;
    return ret;
}
```

类欧几里得

- $m = \lfloor \frac{an+b}{c} \rfloor$.
- $f(a,b,c,n) = \sum_{i=0}^{n} \lfloor \frac{ai+b}{c} \rfloor$: 当 $a \geq c$ or $b \geq c$ 时，$f(a,b,c,n) = (\frac{a}{c})n(n+1)/2 + (\frac{b}{c})(n+1) + f(a \bmod c, b \bmod c, c, n)$；否则 $f(a,b,c,n) = nm - f(c, c-b-1, a, m-1)$。
- $g(a,b,c,n) = \sum_{i=0}^{n} i \lfloor \frac{ai+b}{c} \rfloor$: 当 $a \geq c$ or $b \geq c$ 时，$g(a,b,c,n) = (\frac{a}{c})n(n+1)(2n+1)/6 + (\frac{b}{c})n(n+1)/2 + g(a \bmod c, b \bmod c, c, n)$；否则 $g(a,b,c,n) = \frac{1}{2}(n(n+1)m - f(c, c-b-1, a, m-1) - h(c, c-b-1, a, m-1))$。
- $h(a,b,c,n) = \sum_{i=0}^{n} \lfloor \frac{ai+b}{c} \rfloor^2$: 当 $a \geq c$ or $b \geq c$ 时，$h(a,b,c,n) = (\frac{a}{c})^2 n(n+1)(2n+1)/6 + (\frac{b}{c})^2(n+1) + (\frac{a}{c})(\frac{b}{c})n(n+1) + h(a \bmod c, b \bmod c, c, n) + 2(\frac{a}{c})g(a \bmod c, b \bmod c, c, n) + 2(\frac{b}{c})f(a \bmod c, b \bmod c, c, n)$；否则 $h(a,b,c,n) = nm(m+1) - 2g(c, c-b-1, a, m-1) - 2f(c, c-b-1, a, m-1) - f(a,b,c,n)$。

```cpp
template<typename T>
T lintp(const vector<int>& x, const vector<T>& y, int k) {
    T ans = 0;
    for (int i = 0; i < x.size(); ++i) {
        if (k == x[i])
            return y[i];
        T u = 1, v = 1;
        for (int j = 0; j < x.size(); ++j) {
            if (i == j) continue;
            u *= (k - x[j]);
            v *= (x[i] - x[j]);
        }
        ans += y[i] * u / v;
    }
    return ans;
}
```

中国剩余定理

```cpp
int CRT(std::vector<int> a, std::vector<int> r) {
    long long n = 1, ans = 0, k = a.size();
    for (int i = 0; i < k; ++i) n = n * r[i];
    for (int i = 0; i < k; ++i) {
        long long m = n / r[i], b, y;
        exgcd(m, r[i], b, y);
        ans = (ans + a[i] * m % n * b % n + n) % n;
    }
    return ans;
}
```

各种筛

筛法	场景	效率
Min-25 筛	$f(p)$ 是一个关于 p 的多项式，$f(p^c)$ 能快速求	$O(\frac{n^{\frac{3}{4}}}{\log n})$
PN 筛	找一个好求前缀和的积性函数 $g()$ 在 p 处 $f(p) = g(p)$	$O(\sqrt{n})$
杜教筛	找一个 $g()$ 使得 $f * g()$ 好求前缀和	$O(n^{\frac{2}{3}})$

Min-25 筛

```
namespace Min_25{
    bool INI = 0;
    int M = 0, sqn = 0;
    std::vector<int> primes;
    Z (*fp)(int, int);  // f(p^c)
    Z fp_single(int p, int c) {
        return power(Z(p), c);
    }
    Z sgp(int n, int c) {
        if (c == 0) return Z(n);
        if (c == 1) return Z(n+ 1) * n / 2;
        if (c == 2) return Z(2*n+1) * (n+1) * n / 6;
        // if (c == 3) return Z(n) * n * (n+1) * (n+1) / 4;
        // if (c == 4) return Z(n) * (n+1) * (2*n+1) * (3*n*n%P + 3*n - 1) / 30;
    }
    struct Gpoly{
        int M, sqn, a;
        std::vector<Z> F_prime1, F_prime2;
        Gpoly(int scale, int a, int c): M(scale), a(a) {
            sqn = sqrt(M) + 3;
            F_prime1.resize(sqn), F_prime2.resize(sqn);

            int st = M, ed = 1;
            for (int i = 1; st >= ed; st = min(M / ++i, st - 1)) {
                (*this)[st] = sgp(st, c) - 1;
            }
            for (int j = 1; j < primes.size(); ++j) {
                st = M, ed = primes[j] * primes[j];
                for (int i = 1; st >= ed; st = min(M / ++i, st - 1)) {
                    (*this)[st] -= fp_single(primes[j], c) * ((*this)[st/primes[j]] - (*this)[primes[j-1]]);
                }
            }
        }
        void seta(int aa) {a = aa;}
        Z & operator [](int idx) {
            if (idx < sqn) return F_prime1[idx];
            return F_prime2[M/idx];
        }
    };

    std::vector<Gpoly> Gp;
    Z G(int x){
        Z res = 0;
        for (auto &g: Gp) res += g[x] * g.a;
        return res;
    }
    void init(int scale) {
        if (M < scale) INI = 0;
        M = scale, sqn = sqrt(M) + 5;

        if (!INI) {
            INI = 1;
            std::vector<signed> vis(sqn);
            primes.resize(1, 1);
            for (int i = 2; i < sqn; ++i) {
                if (!vis[i]) primes.push_back(i);
                for (int j = 1; j < primes.size(); ++j) {
                    auto prm = primes[j];
                    if (prm * i >= sqn) break;
                    vis[i * prm] = 1;
                    if (i % prm == 0) break;
                }
            }

        }

        Gp.clear();
        Gp.emplace_back(scale, N, 0);
        // Gp.emplace_back(scale, a, c); // a*p^c
    }
    Z seive(int n, int k) {
        Z res = G(n) - G(primes[k-1]);
        for (int i = k; i < primes.size() && primes[i] * primes[i] <= n; ++i) {
            Z fpj = fp(primes[i], 1), fpj1;
            for (int j = 1, pc = primes[i]; pc * primes[i] <= n; ++j, pc *= primes[i], fpj = fpj1) {
                fpj1 = fp(primes[i], j + 1);
                res += fpj * seive(n/pc, i + 1) + fpj1;
            }
        }
        return res;
    }
    Z S_f(int n) {
        return seive(n, 1) + 1;
    }
    void seta(const vector<int>& a) {for (int i = 0; i < Gp.size() && i < a.size(); ++i) Gp[i].seta(a[i]);}
}
```

PN 篩 (Powerful Number)

```
// f = (g*h); f(p) = g(p) + h(p) [with f(p) = g(p)] -> h[p] = 0.
namespace PowerfulNumber{
    bool INI = 0;
    std::vector<int> primes;
    std::vector<std::vector<Z> > h;
    int M;
    Z (*fp)(int, int);  // f(p^c)
    Z (*gp)(int, int);  // g(p^c)
    Z (*S_g)(int);      // preffix sum of g()

    // if lack of a formula of h(i, j), you need to set f() and g() beforehand
    void init(int scale) {
        if (M < scale) INI = 0;
        M = scale;
        int n = std::sqrt(M)+10;
        if (!INI) {
            INI = 1;
            primes.resize(0);
            std::vector<signed> vis(n);
            for (int i = 2; i < n; ++i) {
                if (!vis[i]) primes.push_back(i);
                for (auto prm: primes) {
                    if (prm * i >= n) break;
                    vis[i * prm] = 1;
                    if (i % prm == 0) break;
                }
            }
            h.resize(primes.size(), std::vector<Z>((int)(log2(M))+1));
        }

        // get the function h() (with f() and g() set or with formula of h())
        for (int i = 0; i < h.size(); ++i) {
            int pp = primes[i] * primes[i];
            if (pp > M) break;
            h[i][0] = 1;
            for (int j = 2; pp <= M && j < h[i].size(); ++j, pp *= primes[i]) {
                Z sgh = 0;
                for (int k = 1, xp = primes[i]; k <= j; ++k, xp *= primes[i]) {
                    sgh += gp(primes[i], xp) * h[i][j-k];
                }
                h[i][j] = fp(pp, j) - sgh;
                // h[i][j] = Z(j-1) * (pp * primes[i] % P - pp % P);
                // h[i][j] = Z(-pp) / (j * (j-1));
                // [a formula of h(i, j)], better faster than log, this example can be optimized to O(1).
            }
        }
    }

    // assistance func to get the sum
    Z PN_sieve(int n, int flr, Z hd) {
        Z res = S_g(n)*hd;
        for(int i = flr+1; i < primes.size(); ++i) {
            int prm=primes[i], k=1;
            int val=n/prm, pk=prm;

            if(val < prm) break;
            while(val >= prm) {
                val /= prm;
                pk *= prm;
                ++k;

                res += PN_sieve(val, i, hd*h[i][k]);
            }
        }
        return res;
    }
    // func to get the sum
    Z getsumf(int n) {
        return PN_sieve(n, -1, 1);
    }
}
```

杜教筛

```
// g(1)S(n) = \sum_{i=1}^{n}(f*g)(i) - \sum_{i=2}^{n}g(i)S(n/i)
namespace dujiaoshai{
    vector<Z> smallS;
    map<int, Z> bigS;

    void Set_smallS(int);
    Z (*S_fg)(int);
    Z (*S_g)(int);
    Z getS(int n) {
        if (n < smallS.size())  return smallS[n];
        else if (bigS.count(n)) return bigS[n];
        Z res = S_fg(n);
        for (int l = 2, r; l <= n; l = r + 1) {
            r = n / (n / l);
            res -= (S_g(r) - S_g(l-1)) * getS(n/l);
        }
        // res /= S_g(1);
        return (bigS[n] = res);
    }
}

void dujiaoshai::Set_smallS(int siz) {
    int n = pow(siz, 0.67);
    smallS.assign(n, 0);
    smallS[1] = 1;
    vector<signed> vis(n), primes;
    for (int i = 2; i < n; ++i) {
        if (!vis[i]) {
            primes.push_back(i);
            smallS[i] = i - 1;
        }
        for (auto &prm: primes) {
            if (i * prm >= n) break;
            vis[i * prm] = 1;
            if (i % prm == 0) {
                smallS[i * prm] = smallS[i] * prm;
                break;
            }
            smallS[i * prm] = smallS[i] * (prm - 1);
        }
        smallS[i] = smallS[i-1] + smallS[i]*i;
    }
}
```

多项式

- Poly with NTT

```
std::vector<int> rev;
std::vector<Z> roots{0, 1};
void dft(std::vector<Z> &a) {
    int n = a.size();

    if ((int)(rev.size()) != n) {
        int k = __builtin_ctz(n) - 1;
        rev.resize(n);
        for (int i = 0; i < n; i++) {
            rev[i] = rev[i >> 1] >> 1 | (i & 1) << k;
        }
    }

    for (int i = 0; i < n; i++) {
        if (rev[i] < i) {
            std::swap(a[i], a[rev[i]]);
        }
    }
    if ((int)(roots.size()) < n) {
        int k = __builtin_ctz(roots.size());
        roots.resize(n);
        while ((1 << k) < n) {
            Z e = power(Z(3), (P - 1) >> (k + 1));
            for (int i = 1 << (k - 1); i < (1 << k); i++) {
                roots[2 * i] = roots[i];
                roots[2 * i + 1] = roots[i] * e;
            }
            k++;
        }
    }
    for (int k = 1; k < n; k *= 2) {
        for (int i = 0; i < n; i += 2 * k) {
            for (int j = 0; j < k; j++) {
                Z u = a[i + j];
                Z v = a[i + j + k] * roots[k + j];
                a[i + j] = u + v;
                a[i + j + k] = u - v;
            }
        }
    }
}
void idft(std::vector<Z> &a) {
    int n = a.size();
    std::reverse(a.begin() + 1, a.end());
    dft(a);
    Z inv = (1 - P) / n;
    for (int i = 0; i < n; i++) {
        a[i] *= inv;
    }
}
struct Poly {
    std::vector<Z> a;
    Poly() {}
    Poly(const std::vector<Z> &a) : a(a) {}
    Poly(const std::initializer_list<Z> &a) : a(a) {}
    int size() const {
        return a.size();
    }
    void resize(int n) {
        a.resize(n);
    }
    Z operator[](int idx) const {
        if (idx < size()) {
            return a[idx];
        } else {
            return 0;
        }
    }
    Z &operator[](int idx) {
        return a[idx];
    }
    Poly mulxk(int k) const {
        auto b = a;
        b.insert(b.begin(), k, 0);
        return Poly(b);
    }
    Poly modxk(int k) const {
        k = std::min(k, size());
        return Poly(std::vector<Z>(a.begin(), a.begin() + k));
    }
    Poly divxk(int k) const {
        if (size() <= k) {
            return Poly();
        }
        return Poly(std::vector<Z>(a.begin() + k, a.end()));
    }
    friend Poly operator+(const Poly &a, const Poly &b) {
        std::vector<Z> res(std::max(a.size(), b.size()));
        for (int i = 0; i < (int)(res.size()); i++) {
            res[i] = a[i] + b[i];
        }
        return Poly(res);
    }
    friend Poly operator-(const Poly &a, const Poly &b) {
        std::vector<Z> res(std::max(a.size(), b.size()));
        for (int i = 0; i < (int)(res.size()); i++) {
            res[i] = a[i] - b[i];
        }
        return Poly(res);
    }
    friend Poly operator*(Poly a, Poly b) {
        if (a.size() == 0 || b.size() == 0) {
            return Poly();
        }
        int sz = 1, tot = a.size() + b.size() - 1;
        while (sz < tot) {
            sz *= 2;
        }
        a.a.resize(sz);
        b.a.resize(sz);
        dft(a.a);
        dft(b.a);
        for (int i = 0; i < sz; ++i) {
            a.a[i] = a[i] * b[i];
        }
        idft(a.a);
        a.resize(tot);
        return a;
    }
    friend Poly operator*(Z a, Poly b) {
        for (int i = 0; i < (int)(b.size()); i++) {
            b[i] *= a;
        }
        return b;
    }
    friend Poly operator*(Poly a, Z b) {
        for (int i = 0; i < (int)(a.size()); i++) {
            a[i] *= b;
        }
        return a;
    }
    Poly &operator+=(Poly b) {
        return (*this) = (*this) + b;
    }
    Poly &operator-=(Poly b) {
        return (*this) = (*this) - b;
    }
    Poly &operator*=(Poly b) {
        return (*this) = (*this) * b;
    }
    Poly deriv() const {
        if (a.empty()) {
            return Poly();
        }
        std::vector<Z> res(size() - 1);
        for (int i = 0; i < size() - 1; ++i) {
            res[i] = (i + 1) * a[i + 1];
        }
        return Poly(res);
    }
    Poly integr() const {
        std::vector<Z> res(size() + 1);
        for (int i = 0; i < size(); ++i) {
            res[i + 1] = a[i] / (i + 1);
        }
        return Poly(res);
    }
    Poly inv(int m) const {
        Poly x{a[0].inv()};
        int k = 1;
        while (k < m) {
            k *= 2;
            x = (x * (Poly{2} - modxk(k) * x)).modxk(k);
        }
        return x.modxk(m);
    }
    Poly log(int m) const {
        return (deriv() * inv(m)).integr().modxk(m);
    }
    Poly exp(int m) const {
        Poly x{1};
        int k = 1;
        while (k < m) {
            k *= 2;
            x = (x * (Poly{1} - x.log(k) + modxk(k))).modxk(k);
        }
        return x.modxk(m);
    }
    Poly pow(int k, int m) const {
        int i = 0;
        while (i < size() && a[i].val() == 0) {
            i++;
        }
        if (i == size() || 1LL * i * k >= m) {
            return Poly(std::vector<Z>(m));
        }
        Z v = a[i];
        auto f = divxk(i) * v.inv();
        return (f.log(m - i * k) * k).exp(m - i * k).mulxk(i * k) * power(v, k);
    }
    Poly sqrt(int m) const {
        Poly x{1};
        int k = 1;
        while (k < m) {
            k *= 2;
            x = (x + (modxk(k) * x.inv(k)).modxk(k)) * ((P + 1) / 2);
        }
        return x.modxk(m);
    }
```

```cpp
    Poly mulT(Poly b) const {
        if (b.size() == 0) {
            return Poly();
        }
        int n = b.size();
        std::reverse(b.a.begin(), b.a.end());
        return ((*this) * b).divxk(n - 1);
    }
    std::vector<Z> eval(std::vector<Z> x) const {
        if (size() == 0) {
            return std::vector<Z>(x.size(), 0);
        }
        const int n = std::max((int)(x.size()), size());
        std::vector<Poly> q(4 * n);
        std::vector<Z> ans(x.size());
        x.resize(n);
        std::function<void(int, int, int)> build = [&](int p, int l, int r) {
            if (r - l == 1) {
                q[p] = Poly{1, -x[l]};
            } else {
                int m = (l + r) / 2;
                build(2 * p, l, m);
                build(2 * p + 1, m, r);
                q[p] = q[2 * p] * q[2 * p + 1];
            }
        };
        build(1, 0, n);
        std::function<void(int, int, int, const Poly &)> work = [&](int p, int l, int r, const Poly &num) {
            if (r - l == 1) {
                if (l < (int)(ans.size())) {
                    ans[l] = num[0];
                }
            } else {
                int m = (l + r) / 2;
                work(2 * p, l, m, num.mulT(q[2 * p + 1]).modxk(m - l));
                work(2 * p + 1, m, r, num.mulT(q[2 * p]).modxk(r - m));
            }
        };
        work(1, 0, n, mulT(q[1].inv(n)));
        return ans;
    }
};
```

- FFT

```cpp
namespace FFT { // n_ 是初始的数组长度, 不一定为 2 的幂次; n 是 init 之后的长度, 保证为 2 的幂次长度
    const double PI = acos(-1);
    int rev[1 << 20];
    int init(int n_) {
        int step = 0, n = 1;
        for (; n < n_; n <<= 1) ++step;
        for (int i = 1; i < n; ++i) {
            rev[i] = (rev[i >> 1] >> 1) | ((i & 1) << (step - 1));
        }
        return n;
    }
    void FFT(complex<double> a[], int n, int f) {
        for (int i = 0; i < n; ++i) {
            if (i < rev[i]) std::swap(a[i], a[rev[i]]);
        }
        for (int h = 2; h <= n; h <<= 1) {
            complex<double> wn(cos(f * 2 * PI / h), sin(f * 2 * PI / h));
            for (int i = 0; i < n; i += h) {
                complex<double> w(1, 0), u;
                for (int j = i, k = h >> 1; j < i + k; ++j) {
                    u = a[j + k] * w;
                    a[j + k] = a[j] - u;
                    a[j] = a[j] + u;
                    w = w * wn;
                }
            }
        }
        if (f == -1) {
            for (int i = 0; i < n; ++i) {
                a[i] = {a[i].real() / n, 0};
            }
        }
    }
    void conv(complex<double> a[], complex<double> b[], int n_) { // n_ does not represent the pow of 2.
        int n = init(n_);
        FFT(a, n, 1);
        FFT(b, n, 1);
        for (int i = 0; i < n; ++i) a[i] *= b[i];
        FFT(a, n, -1);
    }
}
```

组合数学

斯特林数

第二类斯特林数

```cpp
struct Stirling2 {
    std::vector<std::vector<Z> > S;
    Stirling2 (int n) {
        S.resize(n+1, vector<Z>(n+1));
        S[0][0] = 1;
        for (int i = 1; i <= n; ++i) {
            for (int j = 1; j <= n; ++j) {
                S[i][j] = S[i-1][j-1] + S[i-1][j] * j;
            }
        }
    }
    std::vector<Z> &operator [](int idx) {return S[idx];}
};


struct Stirling2_oneline {
    std::vector<Z> S;
    int n;
    // S[n][k] = \sum \frac {i^n (-1)^{m-i}}{i! (m-i)!}
    // that is convolution of f(x) = x^n / (x!) and g(x) = (-1)^x / (x!)
    Stirling2_oneline(int _n): n(_n) {
        vector<Z> f(n+1), g(n+1), ifact(n+1);
        g[0] = 1;

        Z fact = 1;
        for (int i = 1; i <= n; ++i) fact *= i;
        ifact[n] = fact.inv();
        for (int i = n-1; i >= 1; --i) {
            ifact[i] = ifact[i+1] * (i+1);
        }

        for (int i = 1; i <= n; ++i) {
            f[i] = power(Z(i), n) * ifact[i];
            g[i] = ifact[i] * (i % 2? -1: 1);
        }

        S = (Poly(f) * Poly(g)).modxk(n+1).a;
    }
    Z operator [](int idx) {return S[idx];}
};
```

线性代数

线性基

```cpp
struct LinearBase{
    std::vector<int> LBP;
    LinearBase(): LBP(64) {}
    LinearBase(int _n): LBP(_n) {}

    void insert(int x) {
        for (int i = (int)LBP.size() - 1; i >= 0; --i) {
            if ((x >> i) & 1) {
                if (!LBP[i]) {LBP[i] = x; break;}
                else x ^= LBP[i];
            }
            else continue;
        }
    }
    bool query(int x) {
        for (int i = (int)LBP.size() - 1; i >= 0; --i) {
            if ((x >> i) & 1) x ^= LBP[i];
        }
        return (x == 0);
    }
    int MaxSum() {
        int ret = 0;
        for (int i = (int)LBP.size() - 1; i >= 0; --i) {
            if ((ret >> i) & 1) continue;
            else ret ^= LBP[i];
        }
        return ret;
    }
};
```

博弈论

Sprague-Grundy Theorem (SG 定理), 将多个游戏的 SG 值异或起来就能表示组合起来的胜负状态. 可以归纳法来证明.

图论

LCA

- 倍增

```cpp
void dfs(int u, int fa) {
    pa[u][0] = fa; dep[u] = dep[fa] + 1;
    for (int i = 1; i < SP; ++i) pa[u][i] = pa[pa[u][i - 1]][i - 1];
    for (int& v: G[u]) {
        if (v == fa) continue;
        dfs(v, u);
    }
}

int lca(int u, int v) {
    if (dep[u] < dep[v]) swap(u, v);
    int t = dep[u] - dep[v];
    for (int i = 0; i < SP; ++i) if (t & (1 << i)) u = pa[u][i];
    for (int i = SP-1; i >= 0; --i) {
        int uu = pa[u][i], vv = pa[v][i];
        if (uu != vv) { u = uu; v = vv; }
    }
    return u == v ? u : pa[u][0];
}
```

- 树链剖分

```cpp
namespace HLD{
    const int N = 1e5+3;
    std::vector<std::pair<int, int> > g[N]; // 原本的树
    int idx[N]; // idx[x] -> 点 x 的 dfs 序号
    int ridx[N];// ridx[id] -> dfs 序为 id 的点号
    int dep[N]; // dep[x] -> 点 x 的深度
    int siz[N]; // siz[x] -> 以点 x 为根的树的大小
    int top[N]; // top[x] -> 点 x 所在重链的顶点
    int fa[N];  // fa[x] -> 点 x 的父亲结点
    int son[N]; // son[x] -> 点 x 的重儿子
    int clk = 0;// clk -> dfs 序号

    void init() {
        for (int i = 0; i <= clk; ++i)
            g[i].clear();
        clk = 0;
    }
    int lca(int u, int v) {
        while (top[u] != top[v]) {
            if (dep[top[u]] < dep[top[v]]) std::swap(u, v);
            u = fa[top[u]];
        }
        if (dep[u] > dep[v]) std::swap(u, v);
        return u;
    }
    void HLDinit(int rt) {
        std::function<void(int, int)> predfs1 = [&](int u, int f) {
            siz[u] = 1;
            int &maxs = son[u] = -1;
            for (auto &[to, w]: g[u]) {
                if (to == f) continue;
                fa[to] = u, dep[to] = dep[u] + 1;
                predfs1(to, u);
                siz[u] += siz[to];
                if (maxs == -1 || siz[to] > siz[maxs]) maxs = to;
            }
        };
        std::function<void(int, int)> predfs2 = [&](int u, int tp) {
            top[u] = tp;
            idx[u] = ++clk, ridx[clk] = u;
            if (son[u] != -1) {
                predfs2(son[u], tp);
            }
            for (auto &[to, w]: g[u]) {
                if (to != fa[u] && to != son[u]) predfs2(to, to);
            }
        };
        predfs1(rt, -1);
        predfs2(rt, rt);
    }

    void add(int u, int v, int w = 0) {
        g[u].emplace_back(v, w);
        g[v].emplace_back(u, w);
    }
}
```

连通性

tarjan

```cpp
const int N = 1e5 + 3;
std::vector<int> g[N];
int dfn[N], dfnx = 0; // dfs 序
int low[N]; // 最高点
int col[N], colx = 0; // 连通块标号
int stk[N], stktp = 0; // 当前链
bool instk[N];
void tarjan(int u, int f) {
    dfn[u] = low[u] = ++dfnx;
    stk[++stktp] = u;
    for (auto to: g[u]) {
        // if (to == f) continue;
        if (dfn[to]) {if (instk[to]) low[u] = std::min(dfn[to], low[u]);}
        else tarjan(to, u), low[u] = std::min(low[u], low[to]);
    }
    if (low[u] == dfn[u]) {
        ++colx;
        while (stk[stktp] != u) instk[stk[stktp]] = 0, col[stk[stktp--]] = colx;
        instk[stk[stktp]] = 0, col[stk[stktp--]] = colx;
    }
}
```

树

虚树

用栈处理形式上与笛卡尔树类似, 按顺序后处理右链部分

```cpp
namespace VirtualTree{
    const int N = 3e5+3;
    std::vector<std::pair<int, int> > g[N]; // 原本的树
    int idx[N]; // idx[x] -> 点 x 的 dfs 序号
    int ridx[N];// ridx[id] -> dfs 序为 id 的点号
    int dep[N]; // dep[x] -> 点 x 的深度
    int siz[N]; // siz[x] -> 以点 x 为根的树的大小
    int top[N]; // top[x] -> 点 x 所在重链的顶点
    int fa[N];  // fa[x] -> 点 x 的父亲结点
    int son[N]; // son[x] -> 点 x 的重儿子
    int clk = 0;// clk -> dfs 序号

    void init() {
        for (int i = 0; i <= clk; ++i)
            g[i].clear();
        clk = 0;
    }
    int lca(int u, int v) {
        while (top[u] != top[v]) {
            if (dep[top[u]] < dep[top[v]]) std::swap(u, v);
            u = fa[top[u]];
        }
        if (dep[u] > dep[v]) std::swap(u, v);
        return u;
    }
    void HLDinit(int rt) {
        std::function<void(int, int)> predfs1 = [&](int u, int f) {
            siz[u] = 1;
            int &maxs = son[u] = -1;
            for (auto &[to, w]: g[u]) {
                if (to == f) continue;
                fa[to] = u, dep[to] = dep[u] + 1;
                predfs1(to, u);
                siz[u] += siz[to];
                if (maxs == -1 || siz[to] > siz[maxs]) maxs = to;
            }
        };
        std::function<void(int, int)> predfs2 = [&](int u, int tp) {
            top[u] = tp;
            idx[u] = ++clk, ridx[clk] = u;
            if (son[u] != -1) {
                predfs2(son[u], tp);
            }
            for (auto &[to, w]: g[u]) {
                if (to != fa[u] && to != son[u]) predfs2(to, to);
            }
        };
        predfs1(rt, -1);
        predfs2(rt, rt);
    }

    int stk[N]; // 栈
    bool isl[N];
    std::vector<int> vg[N]; // 虚树
    std::vector<int> lst;   // 当前虚树中的点
    void buildvt(std::vector<int> pt, int rt = 0) {
        for (auto x: lst) vg[x].clear(), isl[x] = 0;
        lst.clear();
        sort(pt.begin(), pt.end(), [](int u, int v) {return idx[u] < idx[v];});

        int tp = 0;
        stk[++tp] = rt; lst.push_back(rt);
        for (auto u: pt) {
            if (u != rt) {
                int ll = lca(u, stk[tp]);
                if (ll != stk[tp]) {
                    while (idx[ll] < idx[stk[tp-1]])
                        vg[stk[tp-1]].push_back(stk[tp]), --tp;
                    vg[ll].push_back(stk[tp]);
                    if (idx[ll] > idx[stk[tp-1]]) stk[tp] = ll, lst.push_back(ll);
                    else --tp;
                }
                stk[++tp] = u, lst.push_back(u);
            }
            isl[u] = 1;
        }
        while (tp > 1) {
            vg[stk[tp-1]].push_back(stk[tp]);
            --tp;
        }
    }

    void add(int u, int v, int w = 0) {
        g[u].emplace_back(v, w);
        g[v].emplace_back(u, w);
    }
}
```

计算几何

二维几何：点与向量

```
#define y1 yy1
#define nxt(i) ((i + 1) % s.size())
typedef double LD;
const LD PI = 3.14159265358979323846;
const LD eps = 1E-10;
int sgn(LD x) { return fabs(x) < eps ? 0 : (x > 0 ? 1 : -1); }
struct L;
struct P;
typedef P V;
struct P {
    LD x, y;
    explicit P(LD x = 0, LD y = 0): x(x), y(y) {}
    explicit P(const L& l);
};
struct L {
    P s, t;
    L() {}
    L(P s, P t): s(s), t(t) {}
};

P operator + (const P& a, const P& b) { return P(a.x + b.x, a.y + b.y); }
P operator - (const P& a, const P& b) { return P(a.x - b.x, a.y - b.y); }
P operator * (const P& a, LD k) { return P(a.x * k, a.y * k); }
P operator / (const P& a, LD k) { return P(a.x / k, a.y / k); }
inline bool operator < (const P& a, const P& b) {
    return sgn(a.x - b.x) < 0 || (sgn(a.x - b.x) == 0 && sgn(a.y - b.y) < 0);
}
bool operator == (const P& a, const P& b) { return !sgn(a.x - b.x) && !sgn(a.y - b.y); }
P::P(const L& l) { *this = l.t - l.s; }
ostream &operator << (ostream &os, const P &p) {
    return (os << "(" << p.x << "," << p.y << ")");
}
istream &operator >> (istream &is, P &p) {
    return (is >> p.x >> p.y);
}

LD dist(const P& p) { return sqrt(p.x * p.x + p.y * p.y); }
LD dot(const V& a, const V& b) { return a.x * b.x + a.y * b.y; }
LD det(const V& a, const V& b) { return a.x * b.y - a.y * b.x; }
LD cross(const P& s, const P& t, const P& o = P()) { return det(s - o, t - o); }
// --------------------------------------------
```

字符串

KMP

可以尝试从 $fail$ 树的角度理解

```
bool kmp(string &s, string &p) {
    int n = p.size();
    vector<int> nex(n + 1);
    for (int i = 1, l = 0; i < n; ++i) {
        while (l && p[l] != p[i]) {l = nex[l];}
        if (p[l] != p[i]) nex[i] = 0;
        else nex[i] = ++l;
    }
    n = s.size();
    for (int i = 0, now = 0; i < n; ++i) {
        while (now && s[i] != p[now]) now = nex[now - 1];
        if (s[i] == p[now]) ++now;
        if (now == p.size()) return true;
    }
    return false;
}
```

后缀自动机

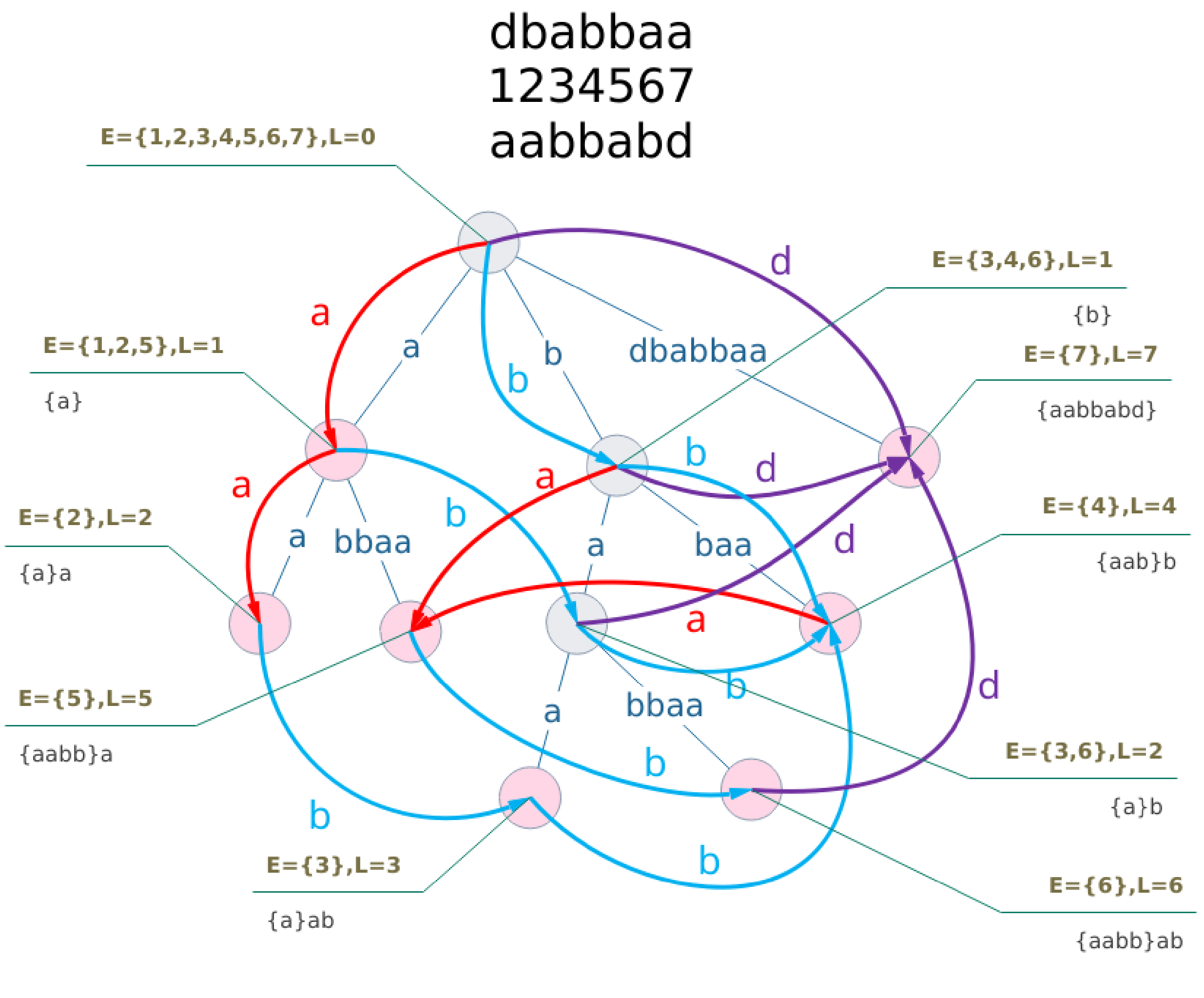

Figure 1: 后缀自动机图解

```
struct SuffixAutomaton {
```

```cpp
    static constexpr int ALPHABET_SIZE = 26;
    int N = 1e5;
    struct Node {
        int len;
        int link;
        // int next[ALPHABET_SIZE];
        std::vector<int> next;
        Node() : len(0), link(0), next(ALPHABET_SIZE) {}
    };
    std::vector<Node> t;
    int cntNodes;
    int extend(int p, int c) {
        if (t[p].next[c]) {
            int q = t[p].next[c];
            if (t[q].len == t[p].len + 1)
                return q;
            int r = ++cntNodes;
            t[r].len = t[p].len + 1;
            t[r].link = t[q].link;
            // std::copy(t[q].next, t[q].next + ALPHABET_SIZE, t[r].next);
            t[r].next = t[q].next;
            t[q].link = r;
            while (t[p].next[c] == q) {
                t[p].next[c] = r;
                p = t[p].link;
            }
            return r;
        }
        int cur = ++cntNodes;
        t[cur].len = t[p].len + 1;
        while (!t[p].next[c]) {
            t[p].next[c] = cur;
            p = t[p].link;
        }
        t[cur].link = extend(p, c);
        return cur;
    }
    SuffixAutomaton(int N_): t(2 * N_), N(N_) {
        cntNodes = 1;
        // std::fill(t[0].next, t[0].next + ALPHABET_SIZE, 1);
        t[0].next.assign(ALPHABET_SIZE, 1);
        t[0].len = -1;
    }
    SuffixAutomaton(string s): t(2 * s.size()), N(s.size()) {
        cntNodes = 1;
        t[0].next.assign(ALPHABET_SIZE, 1);
        t[0].len = -1;

        int p = 1;
        for (auto ch: s) {
            p = extend(p, ch-'a');
        }
    }
};
```

杂项

STL

- copy

```cpp
template <class InputIterator, class OutputIterator>
  OutputIterator copy (InputIterator first, InputIterator last, OutputIterator result);
```